An Application of Fuzzy Logic for Hardware/Software Partitioning in Embedded Systems

Humberto Díaz Pando[1], Sergio Cuenca Asensi[2], Roberto Sepúlveda Lima[1], Jenny Fajardo Calderín[1] and Alejandro Rosete Suárez[1]

[1] School of Informatics Engineering, CUJAE, Havana
Cuba

[2] University of Alicante,
Spain

{hdiazp, sepul, rosete}@ceis.cujae.edu.cu, sergio@dtic.ua.es

Abstract. Hardware/Software partitioning (HSP) is a key task for embedded system co-design. The main goal of this task is to decide which components of an application are to be executed in a general purpose processor (software) and which ones, on a specific hardware, taking into account a set of restrictions expressed by metrics. In last years, several approaches have been proposed for solving the HSP problem, directed by metaheuristic algorithms. However, due to diversity of models and metrics used, the choice of the best suited algorithm is an open problem yet. This article presents the results of applying a fuzzy approach to the HSP problem. This approach is more flexible than many others due to the fact that it is possible to accept quite good solutions or to reject other ones which do not seem good. In this work we compare six metaheuristic algorithms: Random Search, Tabu Search, Simulated Annealing, Hill Climbing, Genetic Algorithm and Evolutionary Strategy. The presented model is aimed to simultaneously minimize the hardware area and the execution time. The obtained results show that Restart Hill Climbing is the best performing algorithm in most cases.

Keywords. Hardware/software co-design, hardware/software partitioning, metaheuristic algorithms.

Aplicación de lógica difusa para el particionado hardware/software en sistemas embebidos

Resumen. El Particionado Hardware/Software (PHS) es una etapa fundamental en el co-diseño de sistemas embebidos. El objetivo principal de esta etapa es decidir qué componentes de la aplicación serían ejecutados en un procesador de propósito general (software) y cuáles en un hardware específico, teniendo en cuenta las restricciones. En los últimos años, se han propuesto diferentes estrategias para resolver el problema PHS, las cuales utilizan en su mayoría algoritmos metaheurísticos. Sin embargo, debido a la diversidad de modelos y métricas utilizadas, decidir qué algoritmo es mejor que otro es un problema abierto. Este artículo presenta los resultados de aplicar lógica difusa en el problema PHS. Esta estrategia es más flexible que muchas de las otras propuestas, ya que es posible aceptar soluciones bastante buenas o rechazar otras que no parezcan buenas. Además en este trabajo se comparan seis algoritmos metaheurísticos: Búsqueda aleatoria, Búsqueda tabú, Recocido simulado, Escalador de colinas, Algoritmo genético y Estrategia evolutiva. El modelo que se presenta está dirigido a minimizar de forma simultánea el área de hardware y el tiempo de ejecución del sistema. Los resultados muestran que el escalador de colinas es el algoritmo que obtiene mejores resultados en la mayoría de los casos.

Palabras clave. Co-diseño hardware/software, particionado hardware/software, algoritmos metaheurísticos.

1 Introduction

Nowadays there are many scenarios where you can find devices that include Embedded Systems (ES) to manage their operation. These systems have three main characteristics [1]: they are (1) single-functioned, (2) tightly constrained and (3) reactive and real-time. The second feature means that the design of an ES is guided by several design metrics. There are many metrics which

can be used to guide the design, e.g., size, performance, cost per unit, flexibility, power consumption, among others; given often the case that for designing a system more than one metric is involved. This implies that the design process itself is complex, since it is necessary to reach a compromise among different metrics.

ES design is fairly complex in most cases; the development of a system requires implementing it on a microprocessor (software component or Sw) and partly on hardware (hardware component or Hw). Traditionally, the design of Hw and the design of Sw are developed separately and in early stages of the design process. This procedure does not ensure compliance with the requirements and generate iterations which increase costs for refining the design. The current trend is to use a unified approach, namely, co-design [2], for the hardware and software components to allow, in addition, verifying the correctness of design, exploring for various possibilities of partitioning without having to go through the costly phase of implementation.

One of the most important stages in the co-design process is the Hardware/Software Partitioning (HSP) [2, 3]. At this stage, the final configuration which the system will adopt must be defined, i.e., a decision about the functional blocks to be implemented in software or in hardware is taken. Usually, this decision is based on the experience of the designer and/or making a brief exploration of the design space. This procedure, in addition to not complying with any methodology, does not ensure an optimal result, since for obtaining the best configuration it is necessary to solve an optimization problem which in most of its formulations is NP-hard [4].

To replace these *ad-hoc* methods, several models [5, 6, 7, 8, 9, 10, 11] have been proposed to reach a solution. These models vary in the applied metrics and in the strategies or algorithms used to solve the optimization problem. In most cases, these models are driven by optimization of a single design metric (area of hardware, execution time or power consumption) and by establishing restrictions over other metrics in order to obtain a desirable solution according to the defined model and to the system interests.

On the other hand, some of these models use exact algorithms to obtain an exact solution to the problem, but the search time increases proportionally to the problem size. Taking into account that in some cases a near optimal solution is considered as good enough, many models use metaheuristics algorithms (Simulated Annealing, Tabu Search, Genetic Algorithms, etc.), which allow to explore the design space to find a good solution achieving an acceptable time for searching a solution. The diversity of used algorithms together with the lack of benchmarks prevents the correct selection of an algorithm that best suits to solve the HSP problem. There are approaches that use other strategies like expert systems combined with the use of fuzzy logic [5] to model the reasoning of the designer and the implicit subjectivity in how this designer solves the problem in practice, modeling variables as fuzzy linguistic variables.

This article presents three contributions. The first contribution involves the proposal of a new HSP model based on the use of the Performance factor metric [12] which is used for finding partitions that take into account two conflicting objectives such as hardware cost (area) and runtime. This approach use fuzzy logic to model the behavior of variables involved in the decision criteria. The second contribution is the application of metaheuristic algorithms with no evidences of prior use for the HSP problem and its comparison with other metaheuristics that have been used actually, yielding interesting results. The third contribution is our study of the feasibility of combining fuzzy logic with metaheuristic algorithms, i.e., applying Soft Computing [13] for solving the HSP problem.

2 Related Work

In recent decades there have been several works that have proposed solutions to HSP. Usually, all the proposals introduce variations in one or more of three basic elements that make up this problem: (1) initial system specification, (2) modeling of the system and (3) searching the solution.

In order to specify the system, it is necessary to define its granularity and an initial implementation [3, 5, 14, 15, 16]. The granularity [15] is the size of a system function block

(instructions, basic blocks, control blocks, functions or procedures) to be considered; this should be taken into account during partitioning.

In addition, the system can be initially implemented in software or hardware. If the system is implemented in software (or hardware) the partitioning strategy is to migrate the functional blocks to the hardware (or software) repeatedly to find the solution which meets the design metrics imposed. The description of the initial implementation can be done by using different languages, preferably a high level language (C, SystemC and VHDL/Verilog), to specify the functions which the system must have.

The model defines how the system will be represented before and during the partitioning process, so at the beginning it is necessary to define a computational model [17] to represent the system.

The most widely used models are control flow graphs, data flow graphs, and call graphs. Many of these computer models are conditioned by the chosen granularity. For example, if a granularity at the function level or at procedure level is selected, the model could be a call graph in which nodes represent system functions and arcs represent calls which one function makes to another. The conclusion reached by the authors in [17] is that the selection of a model depends on a system type. After modeling the system it is necessary to assign for each of the functional blocks the estimated values of the metrics which have been defined. In [5] it is possible to appreciate the diversity of models that can be used in partitioning process.

Finally, to find an optimal solution, or one close to this, it is necessary to define the cost function, constraints, and the algorithm which will solve the optimization problem. In the first two tasks, the design metrics are involved, which are derived from the requirements of the embedded system. There are different metrics to consider for an embedded system designing (runtime, power consumption, size, etc.) [1]; most of the contributions use these metrics in the functions or restrictions depending of the proposed model. In [11] the author makes an abstraction of these metrics considering only three groups: (1) hardware cost, (2) software cost and (3) communication costs between hardware and software blocks. Several proposals use such metrics as hardware area and execution time [6, 8, 18, 19]. In other works the authors combine these metrics or use others like bus utilization and processor [18], execution count of basic blocks [20], the speedup resulting of moving a node from software to hardware, power consumption [21], communication cost [7, 8, 19], and proximity between functions [19].

In relation to the algorithmic aspects, most of the contributions use general-purpose heuristic algorithms. In [14, 22] the authors propose approaches based on the greedy strategy algorithm.

Also, there have been several proposals based on Tabu and Random search algorithms [7, 22, 23, 24], while other solutions are based on Simulated annealing [5, 20, 22, 23, 24], Genetic algorithms [7, 23, 25] and particle swarm optimization (PSO) [9, 10]. López-Vallejo and López [5] and F. Vahid [22] use the Kernighan/Lin algorithm. Several authors [5, 19, 22] use the hierarchical clustering algorithm and Gupta and De Micheli [18] use group migration.

Other studies employ specific-purpose heuristics, such as the proposal of Jigang and Srikanthan [7, 21], while still others use algorithms based on dynamic [21] and linear programming [6, 8, 26].

There are papers that apply fuzzy logic to model the uncertainty of variables; among them there is the proposal of López-Vallejo and López [5], and López *et al.* [27], where the authors suggest an expert system based on fuzzy logic. In this paper, a model defining fuzzy variable sets associated with the characteristics of each node of the application is used. Next, a classification module may obtain a valid solution to the problem. Huang and Kim [28] use a Hybrid Neural Fuzzy System for applying fuzzy logic and neural networks in combination. Zhang *et al.* [29] apply Soft Computing technics, an Evolutionary Negative Selection Algorithm inspired from Artificial Immune System.

On the other hand, there are studies in which the authors compare their proposals with other research, such as the case of Vahid´s work [22] which compares an extension to the Min-Cut algorithm or Kernighan/Lin heuristic with Random Search, Simulated Annealing, Greedy

Improvement, Hierarchical Clustering and clustering followed by a greedy improvement; good results are obtained with the first of these.

López-Vallejo and López [5] compared Kernighan/Lin heuristics with Simulated Annealing, Hierarchical Clustering and an expert system. In [6, 8] the authors modify the algorithm proposed by Madsen *et al*. [26] based on linear programming, then a comparison between both algorithms is done. Jigang *et al.* [7] compare a heuristic algorithm with Simulated Annealing, Tabu Search and Genetic Algorithm. In the work of Wiangtong *et al.* [23] good results are achieved with Tabu Search over Genetic Algorithm and Simulated Annealing.

In summary, most of the contributions are aimed at partitioning models and algorithmic aspects. In the first case we conclude that there is a wide variety of models which apply various metrics, e.g., area occupied and system response time; these metrics are the most frequently used as the objective function and constraints for modeling the problem.

From the point of view of the algorithmic aspect, Table 1 is a summary of the methods used in the works listed above. As it can be seen, in most of the contributions general-purpose heuristic algorithms are used, but there are some that have not yet been evaluated for the HSP problem. Such is the case of Hill Climbing and Evolutionary Strategy which in similar problems have offered good results [30, 31]. Finally, it should be noted that it is difficult to compare algorithms because each author models the problem differently and uses different benchmarks. Thus, it is necessary to perform a unique formulation of the problem in order to compare several algorithms.

3 Hardware/Software Partitioning Model

The HSP model defined in this section considers the following characteristics: granularity, metrics associated with the functional blocks, computational model, representation of the solution, domain of the variables and the cost function.

Let P be the set of functions that make up the program to be partitioned, defined as $P = \{p1,p2,...,p_n\}$ where p_i is a function or program

Table 1. Summary of the contributions to the HSP

	Year							
Algorithms	**1993**	**1997**	**2003**	**2004**	**2006**	**2007**	**2008**	**2010**
Random search		[22]						
Tabu search		[24]					[7]	
Greedy		[22]		[14]			[7]	
Simulated annealing		[22, 24]	[5]					
Genetic algorithm	[20]						[7]	
PSO						[10]	[9]	
Specific heuristics					[21]		[7]	
KL		[22]	[5]					
Group Migration	[18]							
Hierarchical clustering	[22]		[5]					[19]
Dynamic Programming					[21]			
Linear Programming		[26]			[6]		[8]	
Soft Computing Technique			[5]			[28,29]		
Comparison		[22]	[5]				[7]	

method (coarse granularity). Let $G = \{V, E\}$ be the computational model, a call graph, which represents the set of functions or methods that belongs to the program P. In this graph, each vertex v_i belongs to the set $V = \{v1, v2, \ldots, v_n\}$, and it is matched with an element of the set P. The set $E = \{e1, e2, \ldots, e_n\}$ represents dependencies between vertices.

Once the system is represented under this model, values for the metrics are associated to each node. The following metrics are used: software execution time (st_i), occupied hardware area (ha_i), and the hardware execution time (ht_i). Figure 1 shows an example of the process described previously.

In this model, a solution to the partitioning is expressed as a set of binary elements $X = \{x_1, x_2, \ldots, x_n\}$, where the i^{th} element represents the type of implementation assigned to the function p_i; if $x_i = 1$, it means deployment and implementation on Hw, or implementation on Sw otherwise. Given the above stated, it is possible to calculate the hardware area used for designing (A) as:

$$A = \sum_{i=1}^{n} (x_i ha_i) + ArchCost \qquad (1)$$

ArchCost represents a basic architecture cost. Any solution, even pure software implementation, needs a minimal hardware infrastructure (peripherals, memory, processor, etc) which is expressed by ArchCost.

The system execution time (T) is defined by

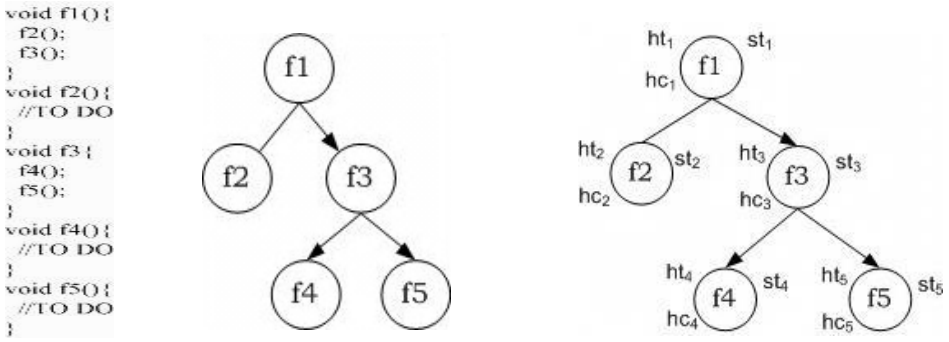
$$T = \sum_{i=1}^{n} \left[(1 - x_i) st_i + x_i ht_i \right] \qquad (2)$$

To take advantage of fuzzy logic, these metrics are modeled as fuzzy sets of the same name as

$$A = \{a, \mu(a)\} \qquad T = \{t, \mu(t)\} \qquad (3)$$

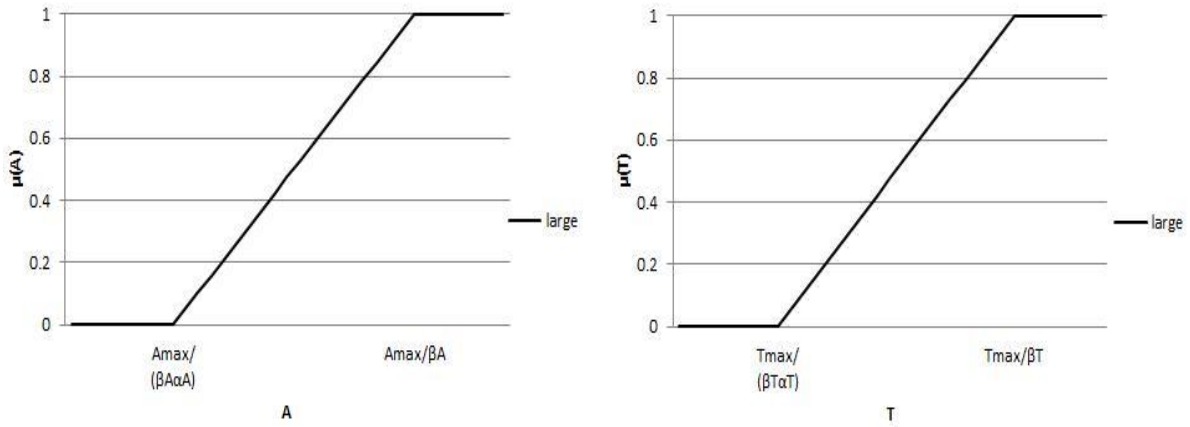
Each one of the elements a or t of the sets will be modeled as a linguistic variable in terms of {*large*} area or {*large*} time. This implies that the pertinence values $\mu(a)$ and $\mu(t)$ of each variable can be obtained applying the Gamma function (see Figure 2).

In Figure 2a, the upper limit of the area variable is defined as A_{max}/β_A, where A_{max} denotes the area required in the case when all blocks are assigned to hardware. The lower limit is defined as a function of the upper limit by $A_{max}/(\beta_A \alpha_A)$. Similarly, the upper and lower limits for the time variable are defined as in Figure 2b; the upper limit of time is defined as T_{max}/β_T, whereas the lower limit is defined as $T_{max}/(\beta_T \alpha_A)$. T_{max} denotes the time consumed by the design in the case when all blocks are assigned to software.



a) b) c)

Fig. 1. Example of set-up process prior to finding solution process



a) Linguistic variable Area b) Linguistic variable Time

Fig. 2. Gamma functions

The use of fuzzy logic makes it possible to have a flexible model in terms of the solution space to be explored, because by varying the lower and upper limits according to the system requirements it is possible to accept quite good solutions or reject other ones which do not seem quite good. In this way the model is more flexible than the classical variant where the upper limits are equal to the lower limits. It is implied that only good solutions are accepted.

Once the initial parameters are established, it is necessary to define the metrics and the objective function which will guide the searching process. In this work the metric is the Performance factor (Pf) proposed in [12], which unifies the hardware area (A) and execution time (T) of the system. It is calculated as Pf = A*T. The goal of using this metric is to obtain a solution that meets both metrics. It is similar to the embedded system designer who executes this task in practice. In this work, the metrics A and T are modeled as fuzzy sets, and Pf is calculated using the pertinence functions defined previously and the function x*y as T-Norm operation. Taking into account the previous elements, the objective function is defined as.

$$Pf = \mu(a) * \mu(t) \quad (2)$$

4 Experiments and Results

As it was discussed above, there is no evidence of a well-defined benchmark that allows an effective comparison between algorithms [7, 11]. To overcome this difficulty, many authors [4, 20, 23, 28] use in their experiments an experimental approach based on graph simulations. In this paper we adopted this approach to conduct the experiments and validate the model.

Our experiments were executed on four systems or problems represented by graphs

Table 2. Characteristics of the generated problems

Count of nodes	Minimum Area	Lower limit	Upper limit	Maximum Area	Minimum Time	Lower limit	Upper limit	Maximum Time
50	10	468	1562	2232	189	282	705	784
100	10	1168	3894	5564	440	618	1547	1719
150	10	1480	4933	7048	648	857	2144	2383
200	10	2258	7529	10757	865	1107	2769	3077

composed by 50, 100, 150 and 200 nodes. The values of software execution time (st_i), hardware area occupied (ha_i) and hardware execution time (ht_i) were generated randomly, as in [4, 20, 23, 28].For the case of software execution time (st_i), values were generated randomly in the range [1, 30]; while for the hardware area (ha_i) values were generated in the range [1,100]. Since the execution time for one node implemented in software is generally greater than in a hardware implementation, the execution time of the node in hardware (ht_i) was generated randomly in the range [1,1/2 * st_i].

For these simulated data, each algorithm was executed 30 times, and in each execution, evaluations of the objective function were made, i.e., for each of the 30 executions, 50000 solutions for each problem were generated. Table 2 summarizes the main features of the problems used in our experiments. As it can be seen, the same *ArchCost* (Minimum area) for all problems was established, and the lower and upper limits were established according to possible user requirements.

In our experiments, we applied Metaheuristics algorithms based on a point (Tabu Search, Hill Climbing, Simulated Annealing), algorithms based on population (Genetic Algorithm, Evolutionary Strategy), and for the aim of comparison we also used the Random Search algorithm.

The algorithms were implemented using the Biciam library [32], which employs a unified model of metaheuristics algorithms. In the case of Simulated Annealing, the parameters used were as follows: *initial temperature* = 15, *final temperature* = 0, and α = 0.93. For Genetic Algorithm the parameters were as follows: initial population of 50 individuals, the truncation factor is 30% of the initial population, while the probability of mutation and crossover was 0.9 for both cases.

Finally, for the Evolutionary Strategy, the count of individuals of the initial population and the truncation factor applied were similar to the Genetic Algorithm, using a mutation probability of 1. It is important to note that we analyzed different values of the parameters of the algorithms, and those that offered best results were chosen.

To assess the quality of the solutions offered by each algorithm, an average of the values of *Pf* obtained in each of the 30 runs is calculated. The use of this measure is intended to provide an idea of how the algorithm finds the best solution. Besides, for each algorithm, the average number of iterations to converge to the best solution was taken.

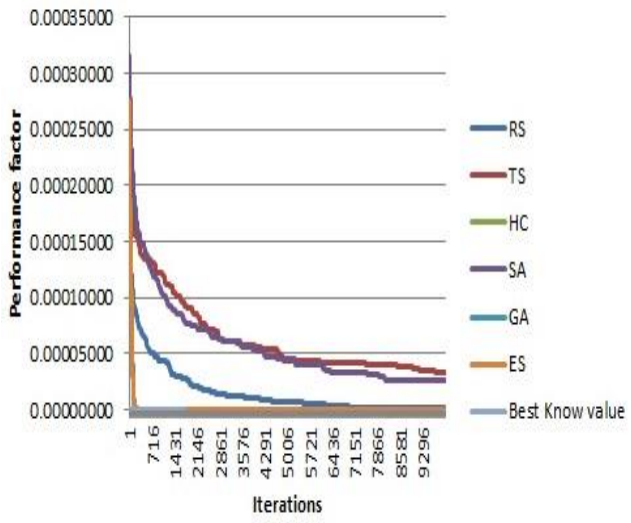
Figure 3 shows the behavior of the algorithms from the average of the results obtained in the evaluation of the goal function for each of the problems discussed; the first 10000 iterations are showed. Notice that the *Best know value* is the best solution obtained by the algorithms, and it is used as a reference for comparing the algorithm's behavior.

As it can be seen in four cases, Tabu Search (TS) and Simulated Annealing (SA) algorithms never reached the results of the other algorithms, although they tended to minimize the objective function. In this way we can see that the Hill Climbing (HC), Genetic Algorithm (GA) and Evolutionary Strategies converge more quickly in most of the problems.

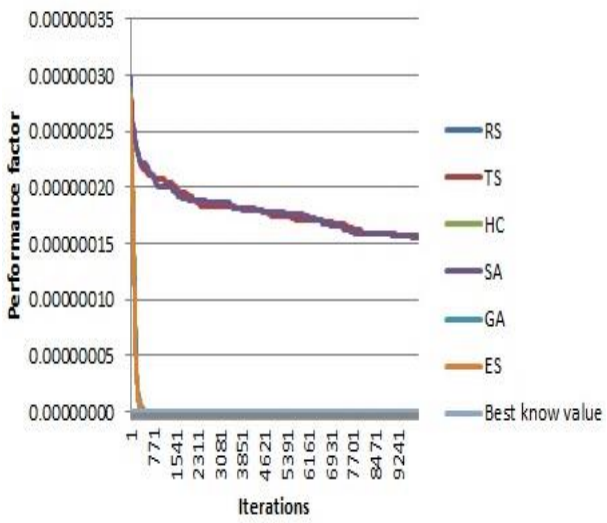
Moreover, as it can be appreciated in the graphs of Figures 3a, 3b y 3d, the Random Search (RS) algorithm tends to reach the minimum value. Taking into account this observation, Figure 4 shows the behavior of the RS algorithm in the last iterations, where it can be seen that this algorithm achieves a better value than the HC (Figure 4a), GA, ES and HC (Figure 4b), HC and GA (Figure 4d).

Table 3 presents a summary of the results for each algorithm, showing the iterations average that reached the minimum value for each of the discussed problems. As one can see, the HC is the algorithm that converges in less iterations than the others for all problems and stands in this value during the rest of iterations, but the quality of its solutions is below the others. This behavior may mean that it reaches a local optimum, which could justify the application of the Restart Hill Climbing (RHC) variant to make use of all the iterations and thus it would reach a better value.

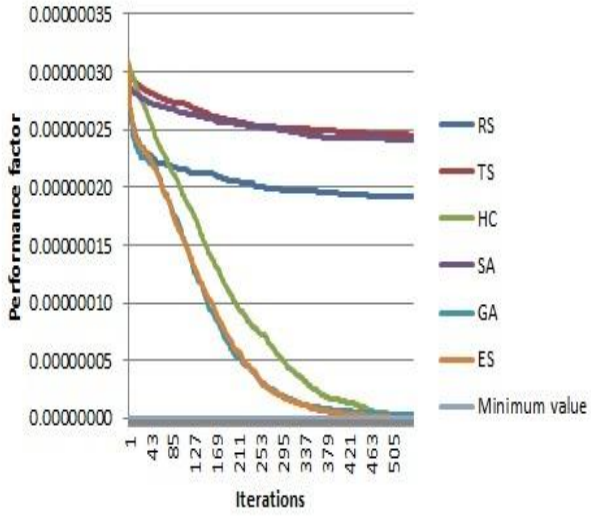
Once the corresponding experiments are fulfilled, we check if the RHC reaches better quality values than the HC and other algorithms (Figure 4).



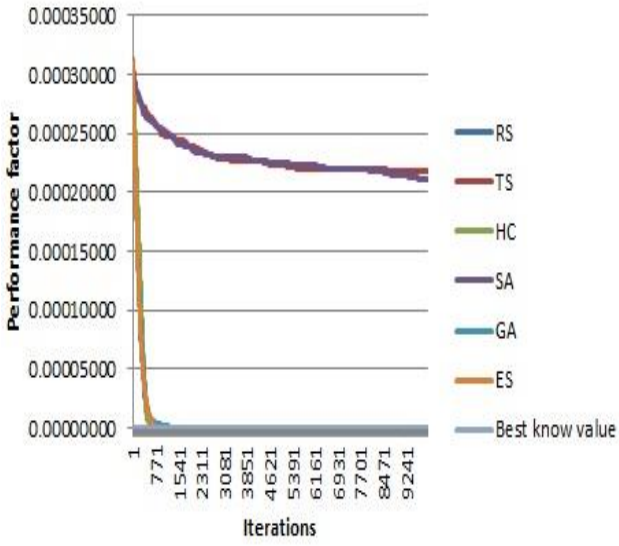
a) 50 nodes



b) 100 nodes



c) 150 nodes



c) 200 nodes

Fig. 3. Average Objective Function per iteration

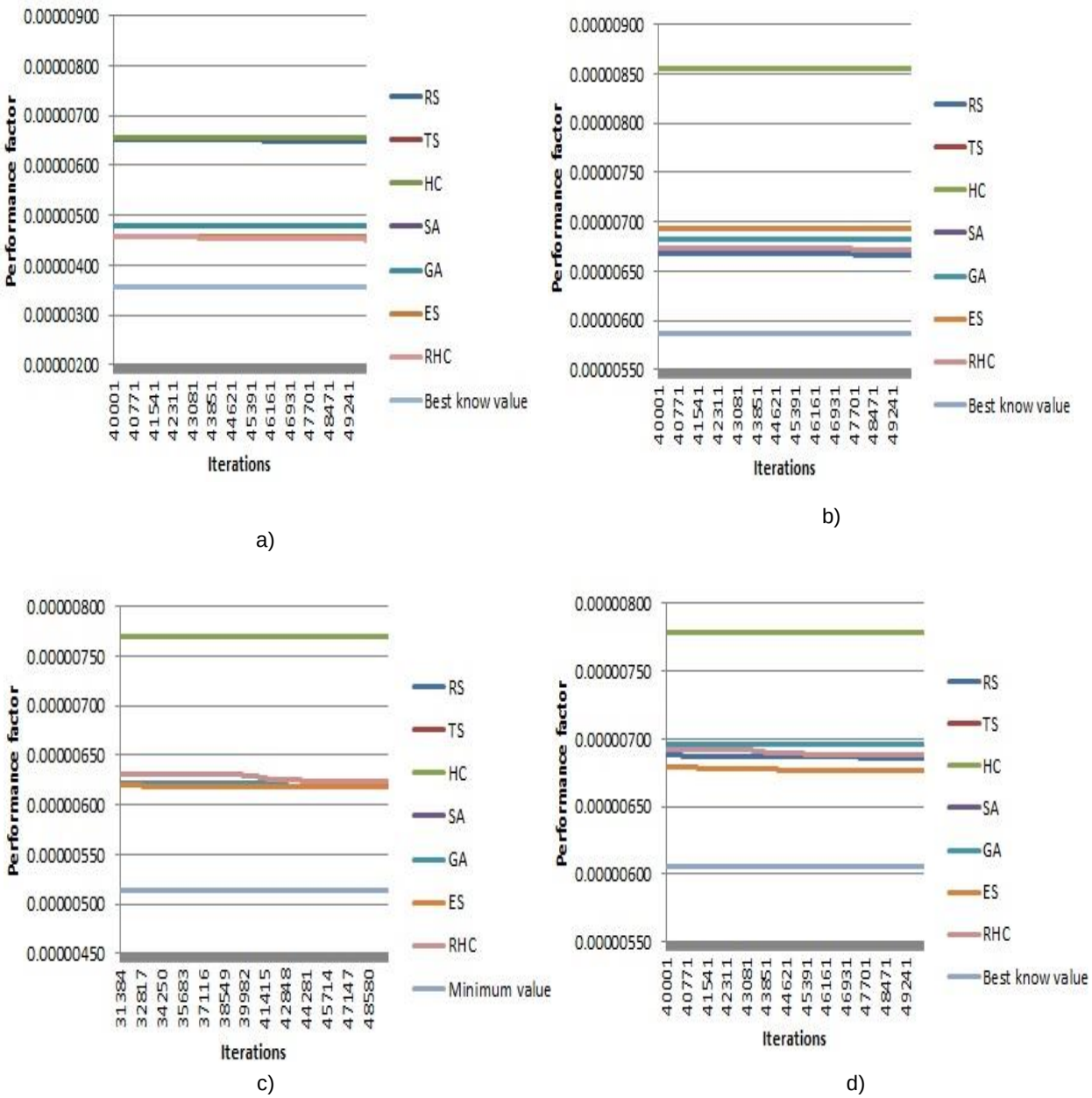


Fig. 4. Behavior of the algorithms over final iterations

Table 3 shows the better value reached by each algorithm with respect to the Performance factor average. In this sense it is important to note that RHC (50 nodes), RS (100 nodes), GA (150 nodes) and ES (150 and 200 nodes) are the algorithms that achieve more quality solutions for each problem. Results similar to these have not been previously reported in related papers, highlighting the fact that the RHC and ES algorithms were not used in any of the approaches mentioned in Section 2.

In some scenarios the embedded system designer could expect that the HSP model returns a list of valid solutions instead of a single solution.

Taking into account the last idea and that in the proposed model there are two conflicting goals in the objective function, we conducted an analysis of the results from the multi-objective point of view over 150 node problem. In this analysis, we calculated the optimal Pareto front [33, 34] for each algorithm, i.e., the non-dominated solutions generated by each of the algorithms.

Moreover, a Unified Pareto Front was generated from the non-dominated solutions obtained above. Finally, we calculated how many

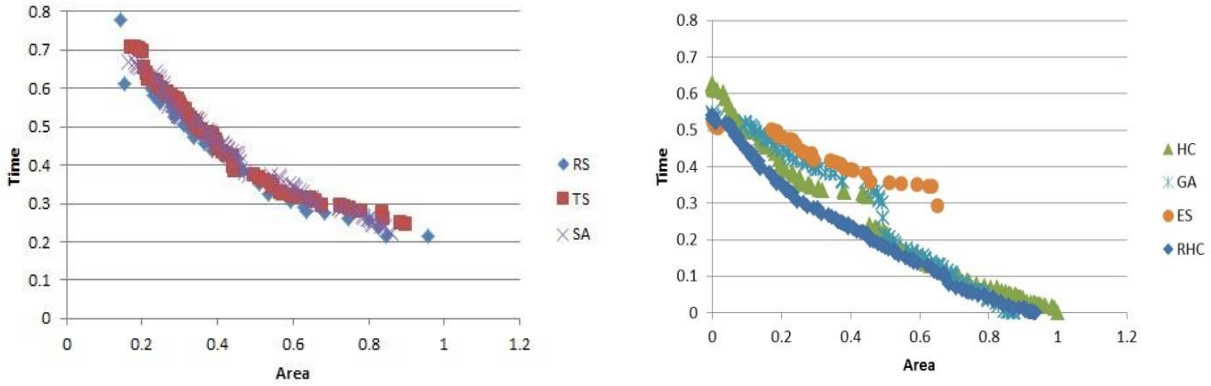
Table 3. Performance factor and Minimum iterations average to converge to minimum value for each algorithm

Count of nodes	Random search		Tabu Search		Hill Climbing	
	Performance factor	Iteration	Performance factor	Iteration	Performance factor	Iteration
50	0.00000649	46.799	0.00506470	49.309	0.00000656	**725**
100	**0.00000665**	48.087	0.13337911	49.253	0.00000856	**12.676**
150	0.13078550	49.152	0.16722894	49.059	0.00000770	**1.428**
200	0.00000686	47.225	0.20198062	49.523	0.00000779	**2.277**
Count of nodes	**Restart hill climbing**		**Simulated annealing**		**Genetic algorithm**	
	Performance factor	**Iteration**	**Performance factor**	**Iteration**	**Performance factor**	**Iteration**
50	**0.00000451**	49.916	0.00353864	49.336	0.00000477	47.388
100	0.00000672	47.638	0.12973347	38.494	0.00000681	42.503
150	0.00000623	49.278	0.16834289	49.992	**0.00000619**	43.023
200	0.00000689	43.818	0.19724766	48.675	0.00000696	36.316
Count of nodes	**Evolutionary strategy**					
	Performance factor	**Iteration**				
50	0.00000458	21.544				
100	0.00000693	22.662				
150	**0.00000619**	32.869				
200	**0.00000677**	44.206				

solutions were provided to the unified front by the algorithms.

Figure 5 shows the Pareto front of each algorithm for 150 node problem. As it can be seen, the RHC algorithm obtained a wider front with a good distribution of the solutions over the entire front. The HC y GA algorithms achieved solutions in a wider front, but in some portions of the front the solutions were sparser. The RS, TS and SA algorithms, despite of achieving solutions well distributed throughout its optimal front, were more compact and distant from the rest of the algorithms.

Table 4 shows a summary of the analysis of non-dominated solutions generated for each algorithm with respect to the total of unique solutions. In general, the percentage of non-dominated solutions for all algorithms is very low with respect to the total amount of distinct solutions generated. As one can see, RS, TS and SA are the algorithms which generate more unique solutions but only a few of these are non-dominated and none of these are present in the unified front. In the particular case of the RS algorithm, it generates solutions with acceptable quality in almost all the problems and also generates a lot of unique solutions, but its contribution to the unified front is null. The HC algorithm generates a fewer amount of unique solutions, but achieves the major percentage of



a) RS, TS, SA. b) HC, ES, GA, RHC.

Fig. 5. Pareto Front for each algorithm

Table 4. Non-dominated solutions for each algorithm and its contribution to Unified Pareto front

Algorithm	Total of generated solutions	Total of non-dominated solutions	Percentage of non-dominated solutions	Solutions in the Unified front	Percentage of solutions in the Unified front
Random search	367.945	38	0,01	0	0
Tabu search	284.707	70	0,02	0	0
Hill climbing	10.089	108	1,0	2	1,8
Restart hill climbing	237.908	119	0,05	93	83,7
Simulated annealing	284.368	83	0,03	0	0
Genetic algorithm	17.011	90	0,53	10	9
Evolutionary strategy	17.193	35	0,2	6	5,4

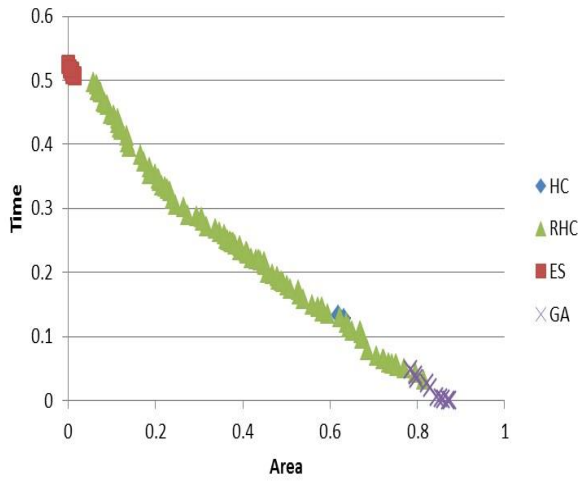


Fig. 6. Unified Pareto Optimal Front

non-dominated solutions, followed by GA and ES. Moreover, these three algorithms contribute with a little percentage of non-dominated solutions to the unified front. It is important to note that HCR was the algorithm that generated more non-dominated solutions, and most of these were present in the unified front.

Figure 6 shows the Unified Pareto Front considering the contributions of each algorithm. As it can be appreciated, the solutions presented in the front are provided by the RHC, GA, ES and HC algorithms. Moreover, the results show three trends: a first group of solutions dominated by area (ES), a second group of solutions dominated by time (GA), a third group covering the center of the front (RHC), including the solutions of the HC algorithm. In other words, the ES algorithm generates solutions with low cost in terms of area occupied by the design but a high impact on execution time, GA operates in the opposite way, and RHC generates solutions more balanced in area and time. The contribution of each algorithm to the Unified Pareto Optimal Front is showed in the last two columns of Table 4. As it can be appreciated, of 111 non-dominated solutions which make up the Unified Front, the highest percentage is provided by the RHC algorithm with 83% of the solutions, with GA, ES and HC providing the rest.

5 Conclusions

This study shows that the RHC algorithm outperforms the GA, ES, HC, RS, SA and TS algorithms over the HSP problem, from the fuzzy logic point of view. To assess the quality of the solutions given by the algorithms, the performance factor metric and a fuzzy approach are introduced which allow to establish thresholds to consider when a good or a bad solution is obtained. The comparison of several metaheuristic algorithms under fuzzy approach for the HSP problem is not present in the related works consulted.

The superiority of the RHC algorithm is given by the quality of the solutions, the amount of non-dominated solutions, and the percentage of these presented in the Pareto front. It is important to highlight that the HCR solutions cover the center of the front, while the solutions of ES and GA are in the end of the front. These algorithms also achieve good quality solutions in many of the problems, like the RS algorithm, but the latter does not give solutions in the Pareto front. Besides, the SA and TS algorithms were the worse, with no solutions in the Pareto front and with discreet values of quality in the generated solutions.

In view of these results, it can be concluded that the algorithm determines the type of solutions obtained (dominated by area or time). This analysis facilitates decision making in selecting the most appropriate algorithm depending on the application constraints.

The comparison was made for different problems with sizes of 50, 100, 150 and 200 nodes. For all instances, values of software runtime (st_i), hardware area (ha_i) and hardware runtime (ht_i) for each node were simulated as in the related work.

References

1. **Vahid, F. & Givargis, T. (2002).** *Embedded System Design: A Unified Hardware/Software Introduction*. New York: Wiley.
2. **De Micheli, G. & Gupta, R.K. (2002).** Hardware-software co-design. In G. De Micheli, R. Ernst, & W. Wolf, (Eds.), *Readings in hardware/software*

co-design (30–44). San Francisco: Morgan Kaufmann Publishers.

3. **Wolf, W. (2003).** A decade of hardware/software codesign. *Computer,* 36(4), 38–43.
4. **Arató, P., Mann, Z.A., & Orbán, A. (2005).** Algorithmic aspects of hardware/software partitioning. *ACM Transactions on Design Automation of Electronic Systems (TODAES),* 10(1), 136–156.
5. **López-Vallejo, M. & López, J.C. (2003).** On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Transactions on Desing Automation of Electronic Systems (TODAES),* 8(3), 269–297.
6. **Wu, J. & Srikanthan, T. (2006).** Low-complex dynamic programming algorithm for hardware/software partitioning. *Information Processing Letters,* 98(2), 41–46.
7. **Jigang, W., Srikanthan, T., & Jiao, T. (2008).** Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design. *Design Automation for Embedded Systems*, 12(4), 345–375.
8. **Wu, J.G., Srikanthan, T., & Zou, G.W. (2008).** New model and algorithm for hardware/software partitioning. *Journal of Computer Science and Technology*, 23(4), 644–651.
9. **Bhattacharya, A., Konar, A., Das, S., Grosan, C., & Abraham, A. (2008).** Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm. *2nd International Conference on Complex, Intelligent and Software Intensive Systems*, Catalonia, Spain, 171–176.
10. **Farmahini-Farahani, A., Kamal, M., Fakhraie, S.M., & Safari, S. (2007).** HW/SW partitioning using discrete particle swarm. *17th ACM Great Lakes symposium on VLSI (GLSVLSI '07),* Stresa-Lago Maggiore, Italy, 359–364.
11. **Mann, Z.Á. (2005).** *Partitioning algorithms for hardware/software co-design*. Ph.D. dissertation, Budapest Univerity of Technology and Economics, Budapest, Hungary.
12. **Mourelle, L.M. & Nedjah, N. (2004).** Efficient cryptographic hardware using the co-design methodology. *International Conference on Information Technology: Coding and Computing (ITCC 2004),* Las Vegas, Nevada, USA, 2, 508–512.
13. **Verdegay, J.L., Yager, R.R., & Bonissone, P.P. (2008).** On heuristics as a fundamental constituent of soft computing. *Fuzzy Sets Systems,* 159(7), 846–855.
14. **Adhipathi, P. (2004).** *Model based approach to hardware/software partitioning of SOC designs*. Master's thesis, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA.
15. **Henkel, J. & Ernst, R. (2001).** An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques*. IEEE Transactions on Very Large Scale Integration Systems*, 9(2), 273–289.
16. **Shaout, A., El-Mousa, A.H., & Mattar, K. (2010).** Models of computation for heterogeneous embedded systems in Electronic Engineering and Computing Technology. *Electronic Engineering and Computing Technology. Lecture Notes in Electrical Engineering*, 60, 201–213.
17. **Cortés, L.A., Eles, P., & Peng, Z. (1999).** *A survey on hardware/software codesign representation models*. Linköping, Sweden: Linköping University.
18. **Gupta, R.K. & De Micheli, G. (1993).** Hardware-software cosynthesis for digital systems*. IEEE Design & Test of Computers*, 10(3), 29–41.
19. **Göhringer, D., Hübner, M., Benz, M., & Becker, J. (2010).** A design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip. *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM),* Charlotte, NC, USA, 259–262.
20. **Ernst, R., Henkel, J., & Benner, T. (1993).** Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of computers,* 10(4), 64–75.
21. **Jigang, W. & Srikanthan, T. (2006).** Algorithmic aspects of area-efficient hardware/software partitioning. *The Journal of Supercomputing*, 38(3), 223–235.
22. **Vahid, F. (1997).** Modifying min-cut for hardware and software functional partitioning. *5th International Workshop on Hardware/Software Co-Design (CODES/CASHE'97),* Braunschweig, Germany, 43–48
23. **Wiangtong, T., Cheung, P.Y.K., & Luk, W. (2002).** Comparing three heuristic search methods for functional partitioning in hardware-software codesign. *Design Automation for Embedded Systems,* 6(4), 425–449.
24. **Eles, P., Peng, Z., Kuchcinski, K., & Doboli, A. (1997).** System level hardware/software partitioning based on simulated annealing and tabu search. *Design Automation for Embedded Systems,* 2(1), 5–32.

25. **Purnaprajna, M., Reformat, M., & Pedrycz, W. (2007).** Genetic algorithms for hardware-software partitioning and optimal resource allocation. *Journal of Systems Architecture: the EUROMICRO Journal,* 53(7), 339–354.
26. **Madsen, J., Grode, J., Knudsen, P.V., Petersen, M.E., & Haxthausen, A. (1997).** LYCOS: the lyngby co-synthesis system. *Design Automation for Embedded Systems*, 2(2), 195–235.
27. **López, M.L., Iglesias, C.A., & López, J.C. (1998).** A knowledge-based system for hardware-software partitioning. *Design, automation & test in Europe*, Paris, France, 914–915.
28. **Huang, Y. & Kim, Y.S. (2007).** Boltzmann Machine Incorporated Hybrid Neural Fuzzy System for Hardware/Software Partitioning in Embedded System Design. *4th International Conference on Modeling Decisions for Artificial Intelligence (MDAI '07)*, Kitakyushu, Japan, 307–317.
29. **Zhang, Y., Luo, W., Zhang, Z., Li, B., & Wang, X. (2008).** A hardware/software partitioning algorithm based on artificial immune principles. *Applied Soft Computing*, 8(1), 383–391
30. **Rosete-Suárez, A., Nogueira-Keeling, A., Ochoa-Rodríguez, A., & Sebag, M. (1999).** Hacia un enfoque general del trazado de grafos. *Revista Iberoamericana de Inteligencia Artificial,* 3(8), 18–26.
31. **Rosete-Suarez, A., Ochoa-Rodriguez, A., & Sebag, M. (1999).** Automatic graph drawing and stochastic hill climbing. *Genetic and Evolutionary Computation Conference*, Orlando, Florida, USA, 2, 1699–1706.
32. **J. Fajardo & A. Rosete. (2011).** Algoritmo multigenerador de soluciones para la competencia y colaboración de generadores metaheurísticos. Revista Internacional de Investigación de Operaciones (RIIO), 1.
33. **Loranca, M.B.B. & Galván, C.G. (2012).** Búsqueda de entorno variable multiobjetivo para resolver el problema de particionamiento de datos espaciales con características poblacionales. *Computación y Sistemas*, 16(3), 335–347.
34. **Gómez, J.C. & Terashima-Marín, H. (2012).** Building general hyper-heuristics for multi-objective cutting stock problems. *Computación y Sistemas,* 16(3), 321–334.

Humberto Díaz Pando is Ph.D. student at Alicante University. He received his Master degree in Applied Informatics at the Faculty of Informatics Engineering, ISPJAE, La Habana, Cuba. His research interests are Hardware-Software Co-design and Partitioning and Embedded Systems for Security.

Sergio A. Cuenca Asensi is associate professor in the Computer Architecture and Technology Department at the University of Alicante, Spain. He received the PhD degree in computer engineering from the University Miguel Herández of Elche, Spain, in 2002. His current research interests include reconfigurable computing, hardware/software codesign and security and dependability in embedded systems.

Roberto Sepúlveda Lima received his M.Sc. and Ph.D. degree at Higher Polytechnic Institute José Antonio Echeverría (CUJAE), Havana, Cuba. His research areas are Artificial Intelligence and Cryptographic Engineering.

Jenny Fajardo Calderín received his M.Sc. degree in Applied Informatics from Higher Polytechnic Institute José Antonio Echeverría (CUJAE), Havana, Cuba. She is Ph.D. student at Granada University. The main subjects of research are associated with basic academic courses and academic activities related to the themes custom Artificial Intelligence.

Alejandro Rosete Suárez received his M.Sc. and Ph.D. degree in informatics from Higher Polytechnic Institute José Antonio Echeverría (CUJAE), Havana (Cuba) in 1995 and 2000, respectively. He joined the Department of Informatics, CUJAE in 1993. Since 2010 is the Head of the Department of Artificial Intelligence and Infrastructure of Informatic Systems (DIAISI). His current research interests include optimization, metaheuristics, soft computing, agent technology and data mining.

Article received on 07/10/2012; accepted on 18/12/2012.